Monads In Javascript

- Ankit Shukla

24 April '15

" Monads Comes with a Curse, once you understand them, You loose the power to explain it to others."
-- Douglas Crockford

Contents

1 Introduction

"Monad is a design pattern used to describe computations as a series of steps".

A monad is really thought as composable computation descriptions or you can use the term computation builder.

Formal Defination :

The formulation of a monad have basically three components :

♥ A *type constructor* M defines how to obtain a corresponding monadic type.

♥ A *Unit* function.that wraps a value of underlying type into a monad.

♥ An *bind* function that chains the operations on a monadic values.

The function and operator must obey the following three laws :

-- Left identity

-- Right identity

-- Associativity

2 Monads and basics :

Use In Imperative Languages :

They are extensively used in pure functional programming languages to manage side effects but can also be used in multiparadigm languages to control complexity and primarily as a way to create an abstraction.

Eg: LINQ (Language-Integrated Query) introduced in Visual Studio 2008 used monads to handle complexity different types of formats of data to do query on and compose the computation.

2.1 Identity Monad

The Identity constructor will serve as the unit function it just creates a monadic value from any value you feed it. To define the identity function just takes an argument and just return the same by transforming it.

```
var identity = function(value) {

  return function() { return value }

};
```

Eg : `identity ('chalkstudio')();` ~ > `'chalkstudio'`

2.2 Bind Function

Bind function just chains the operations on a monadic values.

```
var bind = function(mValue , f) {

  return f(mValue());

};
```

Eg : `function basic(arg){ return "Hi " + arg ;}`

`bind(identity('World'), basic)() ;` ~ > HiWorld

2.2 Maybe Monad

Consider the option type Maybe a, representing a value that is either a single value of type a , or no value at all.

```
data Maybe t = Just t | Nothing                    -- Haskell typeval
```

In JavaScript prespective and how we are going to define it means check the return type and if its undefined or null return Nothing else return the basic computation result that was computed during the computation.

```
var Maybe = function(value) {
 var Nothing = {
  bind: function(fn) { return this; }
 };
 var Something = function(value) {
  return {
   bind: function(fn) { return Maybe(fn.call(this, value)); }
  };
 };
 if (typeof value === 'undefined' || value === null) { return Nothing; }
 return Something(value);
};
```

Bindwill return the value that function returns to but we want a monad to be returned.

```
Maybe.lift = function(f) {
 return function(val) {
  return new Maybe(f(val)); };
}
```

Eg : `var addTen = function(val) { return val + 1;};`

`var maybeAddTen = Maybe.lift(addOne);`

And similarlity we can build the same for arbitary long chains of computation.

4. Monad Laws:

Now as we have the MayBe monad we could just specify the all the laws of Monads. We can define a bind function based on our context or just generically.

4.1 The First Monad Law

```
Maybe(x).bind(f)  ==  Maybe( f (x) )                    -- Left identity
```

4.2 Second Monad Law

```
Maybe(x).bind(function(x) { return x; }) == Maybe(x)    -- Right Identity
```

4.2 Third Monad Law

```
Maybe(x).bind(f).bind(g) == Maybe(x).bind(function(x) {     -- Associativity
  return g(f x));
});
```